目录

第一章 摘要

随着数字化浪潮的推进，Mashup 技术通过融合异质 Web API，为开发者提供了创造独特增值服务的可能。然而，Web API 数量的激增使得开发者在构建 Mashup 应用时面临 API 兼容性及业务逻辑协同的挑战，API 的适配搜索变得耗时且困难。针对这些问题，本文提出了一种创新的 Mashup 组合推荐算法 FAR（Fast API Recommendation），该算法基于静态中心标注法，通过构建高效的离线索引结构和独特的子图评价机制，显著提升了推荐系统的准确率。同时，FAR 算法采用柱状搜索策略，优化了子图生成速度，并避免了过分强调 API 兼容性而忽略实用性的问题，从而提高了推荐效率。此外，本文还提出了根节点优化策略与子图权重优化策略，为 Mashup 推荐领域的研究提供了有益启示。在针对 mashup 数据集的详细实验评估中，FAR 算法在平均运行时间、推荐节点数、精确度（MP）、召回率（MR）以及归一化折扣累积增益（NDCG）等方面均优于当前主流的 API 推荐与图搜索算法，显示出更高的实用性和性能优越性。
关键字：Mashup 技术；Web API；API 推荐系统；静态中心标注法

1.1 Abstract

With the advancement of digitalization, Mashup technology has provided developers with the opportunity to create unique value-added services by integrating heterogeneous Web APIs. However, the surge in the number of Web APIs has posed challenges for developers in ensuring API compatibility and coordinating business logic during the construction of Mashup applications, making the search for compatible APIs time-consuming and difficult. To address these issues, this paper proposes an innovative Mashup combination recommendation algorithm, FAR (Fast API Recommendation), which is based on the static centroid annotation method. By constructing an efficient offline indexing structure and a unique subgraph evaluation mechanism, FAR significantly improves the accuracy of the recommendation system. Furthermore, FAR adopts a columnar search strategy, optimizing the speed of subgraph generation and avoiding the problem of overemphasizing API compatibility while neglecting practicality, thereby enhancing recommendation efficiency. Additionally, this paper also proposes a root node optimization strategy and a subgraph weight optimization strategy, providing valuable insights for research in the field of Mashup recommendation. In detailed experimental evaluations using a Mashup dataset, FAR outperforms current mainstream API recommendation and graph search algorithms in terms of average running time, recommended node count,

precision (MP), recall (MR), and normalized discounted cumulative gain (NDCG), demonstrating its higher practicability and performance superiority.
Keywords: Mashup technology; Web API; API recommendation; Static centrality annotation method.

第二章　引言

2.1　研究背景

随着互联网技术的飞速发展，网络应用和服务呈现出爆炸性增长，推动了 Mashup 技术的诞生与流行。作为一种轻量级的服务组合模式，Mashup 的核心在于将支持 Web API 的应用进行有机组合，进而创造出全新的增值服务，不仅提升了资源利用率，更极大地丰富了网络应用的功能与用户体验[1]。

从发展历程来看，Mashup API 经历了由简单到复杂、由单一到多元的演变。其最初应用主要集中在地图、新闻、社交媒体等领域，例如将地图 API 与房地产数据相结合，生成房产信息地图等[2-3]。随着技术的不断进步和开放 API 数量的增多，Mashup 的应用范围已逐渐扩展至教育、医疗、金融、娱乐等多个行业，成为网络应用开发不可或缺的一部分[4]。

然而，随着越来越多的企业和部门将服务、数据或资源以 API 的形式发布到互联网上，API 数量急剧增加，形成了一个庞大而复杂的 API 服务集合。当开发者尝试将这些 API 组合起来构建 Mashup 应用时，他们常常面临兼容性问题，如不同的认证机制、数据格式、请求/响应协议以及版本差异等，这些都增加了集成的复杂性和风险[5]。

此外，即使找到了技术上兼容的 API，开发者还需考虑它们的业务逻辑是否契合。例如，一个旅游规划应用可能需要集成地图服务、酒店预订、航班查询等多个 API。这些 API 不仅需要能够相互通信，还需要在业务流程上形成闭环，以确保用户体验的流畅性[6-8]。

由于缺乏统一的标准和有效的搜索、评估工具，面对海量的候选 API，开发者在筛选和测试 API 时通常需要花费大量时间和精力，这不仅降低了开发速度，也增加了项目成本。因此，如何高效地在海量 API 中找到兼容且符合业务需求的 API 已成为当前 Mashup 开发领域亟待解决的问题[9-11]。

而在 Mashup API 的研究与实践中，科研人员面临着诸多难点和挑战，其中推荐搜索时间复杂度和推荐精准度尤为突出。

一、推荐搜索时间复杂度高

随着 API 数量的不断增长，如何高效地从海量 API 中搜索并推荐出符合需求的 API 成为了一个重要问题。现有的 API 推荐系统普遍面临搜索时间复杂度高的问题，这在一定程度上制约了其应用效果。目前，主流的 API 推荐方法多采用基于动态规划的斯坦纳树生长算法，然而，在众多的候选 API 中寻找两个节点间的最短路径耗时较长。同时，API 之间的关联性和组合性也增加了搜索推荐的难度，使得搜索过程变得更加耗

时[12]。为提高搜索效率，研究者们提出了各种优化方法，如基于语义的 API 搜索[13]、利用机器学习技术改进搜索算法[8,12]等。然而，这些方法在实际应用中面临 API 的描述、功能等信息多样且复杂的问题，导致搜索算法需要处理大量的数据和信息，需要进一步的研究和改进[14]。

二、推荐精准度较低

除了面临搜索时间复杂度高的问题外，Mashup API 推荐还面临着推荐精准度低的难题。鉴于每个功能通常与海量的 API 相关联，实现特定功能组合的 API 组合方案繁多，导致现有的 API 推荐系统常常难以精确地推荐出符合开发者实际需求的 API[2]。这种情况不仅严重拖慢了开发者的开发进度，还对 Mashup 应用的整体质量和用户体验产生了不良影响[15]。为了提高 API 推荐的精准度，研究者们采用了多种方法，如基于用户行为的推荐、基于内容的推荐以及协同过滤推荐等。虽然这些方法在一定程度上提升了推荐的准确性，但仍然面临着诸如冷启动、数据稀疏性等问题。另外，有些学者尝试使用关键词搜索方法，根据 API 组合间的兼容性来推荐 API 组合，但却未能充分考虑所推荐 API 的实际应用场景和意义[16-17]。这就导致了即便 API 组合兼容度较高，实际使用中却并不符合开发者预期，命中率较低。

因此，在 Mashup API 研究领域，如何在显著提升推荐效率的同时，进一步提高 API 推荐的精准度，确保推荐的 API 既高度兼容又能精准匹配开发者的实际需求，仍是一个亟待解决的关键问题。

为解决这一挑战，本文提出了一种创新的算法。该算法不仅深入考虑了 API 之间的兼容性以及实际业务逻辑，而且通过优化算法结构，显著降低了时间复杂度。通过实施这一算法，我们能够在保证运行时间大幅缩短的同时，显著提高 API 推荐的准确性，为用户推荐出既符合实际业务需求又高度兼容的 API 组合。

2.2 文献综述

2.2.1 基于关键词的 API 推荐

Web API 推荐作为近年来的研究热点，其核心目标在于根据用户的特定功能需求，为开发者精准推荐合适的 Web API。这种推荐方法对于提升开发者的工作效率、增强软件质量和兼容性具有至关重要的作用。随着 Web API 在各类应用中的广泛运用，如何确保推荐的准确性和高效性成为了该领域研究的关键所在。

2021 年，学术界对 Web API 推荐的研究取得了显著进展。其中，一篇论文深入探讨了基于关键词驱动的推荐方法，强调了全面覆盖 API 的重要性。该方法通过精准捕捉关键词与 API 之间的关联，为开发者提供了更为全面且精准的推荐结果。实验报告表明，该方法在有效性和效率上都展现出了卓越的性能[18]。

同年，另一项研究则提出了一个综合性的个性化推荐框架。该框架充分利用了开发者的历史调用记录，并结合认知知识，为开发者提供了更为贴合其需求的推荐服务。该论文进一步提出了两种个体模型和集成模型，通过联合矩阵分解和认知挖掘等技术手段，实现了对现有推荐方法的性能提升[19]。

进入 2022 年，Web API 推荐技术的研究继续深入。Lianyong 等人提出了一种针对 Web API 关联图的 War 方法，该方法将 Web API 的发现、验证和选择操作融为一体，为开发者提供了一个更为高效且便捷的推荐流程[9]。同时，Wenwen 等人也提出了一种新颖的高效 Web API 推荐方法（E-War），利用局部敏感哈希技术为开发者推荐理想的 Web API 网络[20]。此外，Junwu 等人则介绍了一种基于集成的方法，该方法通过多特征模型整合了机器学习和深度学习的优势，为 Open APIs 的推荐提供了强大的技术支持[21]。

同年, Hao Wu 等人提出了一种基于多模型融合和多任务学习的神经框架（MTFM），旨在解决 Web API 数量增长导致的 Mashup 创作中 API 选择困难的问题。该框架通过结合语义分析和特征交互，有效提升了 API 推荐的准确性和效率，并通过实验验证了其在推荐性能上的优势[22]。

到了 2023 年，Web API 推荐技术的研究继续取得新的突破。Huaizhen 等人提出了 Di-rar 方法，旨在检索和推荐用于 Mashup 创建的 Web API 组，以满足软件开发者在创建应用程序时的多样化需求[23]。另一项研究则专注于解决为个性化和兼容的移动应用开发提供 Web API 关联图的挑战，通过创新的技术手段为开发者提供了更为精准和实用的推荐服务[24]。此外，还有一篇论文介绍了一种新的矩阵分解（MF）模型，该模型能够准确预测用户和 Web API 之间的关系，为个性化推荐提供了有力的数据支持[25]。

经过综合对比和分析相关文献，我们可以得出，这些研究均集中于探讨如何通过更高效的 Web API 推荐方法来促进开发者的工作效率与软件质量的提升。这些方法涵盖了从关键词驱动的推荐到利用开发者历史调用记录的个性化推荐框架，旨在通过精确匹配开发者需求与合适的 Web API 来优化开发流程。然而，在实际应用中，特别是面对大规模数据处理时，一些推荐算法显示出了较高的时间复杂度，进而影响了计算效率。具体来说，如基于矩阵分解的方法（例如 MF 模型）或处理复杂 API 关联图所需的方法（如 War 方法），在处理庞大数据集时，其效率受到明显影响。

进一步分析表明，以 API 间兼容性为主要考虑因素的关键词匹配推荐方法在实际应用中可能不足以全面理解开发者的综合需求。虽然此方法推荐的 API 组合有较高兼容性，但由于未充分考虑 API 的具体应用意图和场景，推荐结果可能与开发者的实际需求不完全吻合。这种方法的局限性不仅可能导致推荐系统的命中率下降，还有可能对软件产品的功能性和终端用户的满意度产生负面影响。因此，推荐系统的设计需要

超越简单的兼容性匹配，通过深入分析和理解开发者需求的复杂性和多样性，实现更加精确和个性化的 API 推荐。

2.2.2 组斯坦纳树算法

斯坦纳树问题作为组合优化领域的关键议题，致力于寻求图中权重和最小的最小生成树，对于物流、交通网络设计等领域具有广泛的应用价值。然而，由于其 NP-hard 属性，直接解决斯坦纳树问题极具挑战性。因此，研究者们不断提出启发式算法和近似算法以应对这一难题。

2021 年，一项重要研究通过动态规划等方法深入探讨了顶点无权图中实值图的斯坦纳树问题。该算法无需近似保证，其参数介于 2 和 γ 之间，并在不同场景下均展现出卓越性能，显著超越了当时的技术水平[26]。

2022 年，斯坦纳树问题的研究取得了进一步突破。研究不仅探索了非负边权图中群斯坦纳树（GST）问题的解决方案，还结合了变邻域搜索（VNS）等方法的元启发式框架，展现出优于遗传算法的性能[27]。同年，Shuang 等研究者提出了三种近似算法，成功解决了每个兴趣点（POIs）的斯坦纳树问题，并在寻找概率群斯坦纳树的任务上超越了现有先进方法[28]。此外，Guy 等人针对 DB-GST 问题——即在一个无向图中寻找至少包含群 S 中一个顶点的成本最小的树的问题——提出了一种双标准近似法，在树的成本和叶片比率上均实现了双标准近似[29]。

进入 2023 年，研究者们将斯坦纳树问题的研究扩展到时间图中，为解决社交网络中的时间信息问题提供了新的视角。他们引入的动态规划算法不仅证明了有效解决方案的必要性，还在真实时间网络中验证了其效率和有效性[30]。

2024 年，研究焦点转向社会劳动分工中的多属性群决策问题。研究者提出的多属性群决策方法，结合理论、Ster 点约束和植物生长模拟算法，不仅展现了一种全面的评估决策方法，还通过设计的植物生长模拟算法获得了最优选择。实际案例验证了该方法的有效性和合理性[31]。

尽管先前的研究在斯坦纳树问题上通过动态规划和元启发式方法取得了一定进展，但在处理大规模图数据时，这些方法在计算效率方面仍存在显著不足。动态规划方法虽理论上能确保最优解，但由于其高时间复杂度，实际应用中效率受限；而元启发式方法虽速度快，但难以保证解的全局最优性，且在复杂情境下性能易受影响。因此，对于即时性 Web APIs 推荐，迫切需要一种快速且高效的组斯坦纳树搜索方式以提升效率。

2.2.3 基于图搜索的服务组合方法

基于图的服务组合法，作为一项新兴的研究领域，主要采用图论的理论与方法，旨在探索服务组合问题的解决之道。在现实应用场景中，服务组合往往牵涉众多服务与

用户的多样化需求，如何高效地实现服务的有效组合，以满足用户的需求，成为了研究者们必须面对的重要课题。

2020 年，范国栋及其团队针对服务组合问题，提出了一种创新的基于有向二分图的服务组合图模型。通过精巧地构建服务组合图，并结合最少服务组合查询与 Dijkstra 搜索算法，该团队成功地寻找到了最优服务组合的解决方案。此方法不仅显著提升了服务组合的准确性，还有效降低了计算的复杂性，为解决服务组合问题提供了一种高效的工具[32]。

2021 年，Jiang 等研究人员推出了一款融合知识图谱与协同过滤技术的服务推荐模型。该模型通过深入探索 API 与 Mashup 之间的内在联系，有效地缓解了数据稀疏问题，从而提升了服务推荐的准确度[33]。同年，Shi 及其团队聚焦于 Web 服务网络的嵌入问题，提出了一种预测 Mashup 与 API 间潜在连接的概率主题模型。继而，他们开发了服务图卷积网络（Service-GCN），以学习服务的向量表示，该技术在服务分类与 Mashup 聚类任务中展现了优异的表现[34]。

至 2023 年，Chen 等人创新性地定义了服务-关键词相关图（SKCG），以捕捉服务与关键词之间的关系及服务间的兼容性。他们提出了一种基于关键词的深度强化 Steiner 树搜索方法（K-DRSTS），旨在为 Mashup 创建推荐服务，并通过实际世界数据集的实验，证明了其方法的有效性[35]。

尽管基于图的服务组合方法在解决服务组合问题上取得了令人瞩目的成果，但该方法仍面临一些不容忽视的挑战与局限性。具体而言，构建和管理大规模服务组合图需要庞大的计算资源，并需要借助高效的算法来支持实时的服务组合和推荐。为了进一步提升服务组合的效率，迫切需要设计一种更为高效的图搜索方法，能够快速找到最优的服务组合方案，以实现更为精准和实时的服务组合推荐。

第三章　模型介绍

3.1　基于兼容性的 API 推荐与高效搜索策略

Mashup 技术旨在通过互相兼容的 API，实施一系列多元化的功能，每一种功能均对应着一个庞大的 API 集群。成功地锁定候选功能 API 集群之后，挑选出最适宜的 API 节点成为了关键性的任务。

本文深入探讨了基于兼容性的 API 推荐问题。在 API 组合的过程中，各个 API 并非孤立存在，而是需要相互协作以达成特定功能。选定实现这些功能的 API 节点后，关键在于确保 API 之间的互操作性，并通过适当的连接节点将这些 API 串联起来。这一过程通常涉及在由 mashup 数据集构建的知识图谱中寻找最优的斯坦纳树。

但在斯坦纳树的构建过程中，随着节点数目的激增，搜索空间的复杂度呈现指数级的增长，这对搜索算法的效率提出了更高的挑战。因此，迫切需要设计出一套高效的算法及策略，以在庞大的搜索空间中快速精准地锁定合适的 API 节点，从而优化功能实现和提升系统运行效率。

正是基于这样的需求，本研究致力于通过优化查询策略与子图权重度量方法来解决这一挑战。本文的研究成果主要包括三个方面：

1）构建高效离线索引以加速服务组合搜索：为了显著减少搜索时间，我们构建了离线索引结构，该结构能够在不牺牲准确性的前提下，极大地提升搜索效率。

2）基于优化子图权重和搜索策略的关键 API 选取：通过优化子图权重度量方式与搜索策略，我们能够快速且准确地识别出与用户需求最为匹配的 API，从而确保了 API 组合的准确性和有效性。

3）面向用户场景的 API 子图扩展策略：我们提出了子图的扩展生成策略，这一策略不仅考虑到了 API 的兼容性，还充分考虑了用户的实际应用场景。

3.2　FAR 算法 API 推荐流程

3.2.1　Hub Labeling 离线索引建立

FAR 算法在实现过程中，关键步骤在于计算两点间的最短距离及确定最短路径。然而，当应用于大型知识图谱时，采用传统的在线计算方法会因节点和边的数量巨大而导致计算耗时较长，无法满足实际应用中对于响应速度的高要求。同时，若选择将任意点对之间的距离和路径信息预先存储，虽然可以显著提高查询速度，但这种方式将带来存储空间的急剧增长，特别是在资源有限的情况下，这种策略并不切实际。

为了在时间效率和空间占用之间找到最佳平衡点，本文算法引入中心标注法（Hub Labeling）进行优化处理。中心标签优化是图数据库领域的一种关键技术，其原理在于通过为网络中的一部分关键节点分配“中心”标签，并利用这些中心节点与其他节点的连接关系来加速大型网络中的距离查询。这种方法的成功与否，关键在于如何有效地选择和标记中心节点，以及如何有效利用这些标签来提升查询性能。

近年来，中心标签优化算法在学术界备受瞩目，相关研究持续深入。2020 年的一篇论文提出了一种针对图分析的中心标记方案，通过精确计算 s 和 t 之间的最短路径，有效提升了搜索效率。该研究不仅优化了搜索算法，还引入了中心推动和图缩减技术，大幅减小了索引大小，从而显著降低了存储和计算成本[36]。

进入 2022 年，关于中心标签算法的研究进一步拓展。一篇论文提出了一种创新的 2-Hop 标签算法，专门用于计算最短环路（简称 CSC）。该算法采用动态更新的 CSC 索引，显著提高了查询效率，尤其在处理边插入时展现出卓越的性能[37]。同年，另一篇论文则介绍了一种可定制的中心标签变体，特别适用于路线规划中的边成本问题。该研究不仅提出了一个近似算法来优化平均标签大小，还提供了高效的定制算法，以满足不同场景下的需求[38]。

此外，Wentao 等人在 2022 年提出了一种并行最短距离标签（PSL）方案，专门针对小世界网络进行优化。该方案提出了有效的索引压缩技术，通过显著减小索引大小来提高查询效率。实验结果显示，在十亿规模的图上，该方案表现出高效性能，同时在多核环境中实现了接近线性的加速效果，索引大小减少了高达 94%[39]。

2023 年，一项新的研究突破引起了广泛关注。一篇论文介绍了一种全新的层次化 2-Hop 索引（H2H-Index），特别适用于长距离查询。该算法基于距离保持图构建 H2HL-Index，相较于现有技术，其查询处理速度提升了数个数量级，为大规模网络的高效查询提供了有力支持[40]。

这些研究工作从理论上证明了中心标签优化算法的有效性和优越性，共同推动了中心标签优化在大型知识图谱等图数据库应用领域的发展和应用。

中心标签优化算法的核心组成部分是静态的 HL 索引结构，该结构旨在为图数据库提供离线的索引支持。通过利用这一索引结构，在计算任意两个节点 u 和 v 之间的距离 dist(u, v) 时，无需再频繁地遍历原图，从而极大地提升了计算效率。通过 HL 计算两点间最短距离过程如算法 1 所示：

算法1：建立 HL

输入：节点序号u、v，api间最短距离标签集合L

输出：节点u、v间最短距离

1：$\mathrm{dist}_{(u,v)} \leftarrow +\infty$将$u$、$v$间初始距离设置为正无穷

if $L(u) \cap L(v) = \varnothing$ **do** 如果 L(u) 和 L(v) 的交集为空则返回无穷大

  return $dist_{(u,v)}$

for each $h \in L(u) \cap L(v)$**do** 找两节点间最短距离所经过的中间节点

  if $dist > dist(u,h)+dist(v,h)$ **do** 如果 u 和 v 到标签节点的距离之和小于当前值则进行更新

    $mid \leftarrow h$

    $dist = dist(u,h) + dist(v,h)$

  end if

end for

return $dist_{(u,v)}$

这里，dist(u, v) 代表节点 u 和节点 h 之间的最短距离。dist(u, h) 和 dist(v, h) 以及对应的中心节点 h 都被预先存储在 L(u) 和 L(v) 中。这种预先计算并存储的方式大大加速了查询过程。

为了更精确地刻画节点在图结构中的关键位置，本文借助 SHL 算法，引入介度中心性作为节点的权重，并按照此权重的递减顺序对节点进行排序。

介度中心性作为一种有效的节点重要性度量指标，能够捕捉到节点在图结构中的关键位置。通过计算节点的介度中心性，并将节点按照其介度中心性值进行递减排序，我们能够优先处理那些在网络中起到桥梁作用的节点。这些节点更有可能出现在多对节点之间的最短路径上，因此在建立索引时具有更高的优先级。

具体来说，节点的介度中心性定义为所有节点对之间的最短路径中经过该节点的路径数与所有最短路径数之比。数学表达式如下：

$$\mathrm{bc}(v) = \sum_{s,t \in V \setminus \{v\}} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

其中，σ_{st} 表示从节点 s 到节点 t 的最短路径的数量，而 $\sigma_{st}(v)$ 则表示这些最短路径中经过节点 v 的路径数。

利用介度中心性建立 HL，不仅减少了存储资源的消耗，还提高了搜索效率，使得模型在实际应用中更加高效和可行[41]。建立 HL 算法如算法 2 所示。

算法 2：建立 HL

输入：知识图谱$G = \langle V, E, W \rangle$

输出：G 的静态 HL

$L_0(v) \leftarrow \varnothing$ for $v \in V$

按 bc 值对 V 中节点递减排序，并对节点按照bc值从大到小重新编号

for $i \leftarrow 1$ to n **do**
  $visited[v] \leftarrow 0$ for all $v \in V$
  $d[v_i] \leftarrow 0$ and $d[v] \leftarrow \infty$ for all $v \in V \setminus \{v_i\}$
  $PQ \leftarrow \{v_i\}$ 优先级为$d[v]$的优先级队列
  while $\mathrm{PQ} \neq \emptyset$ **do**
    $u \leftarrow PQ.\mathrm{pull}()$
    $visited[u] \leftarrow 1$
    if $d[u] < \mathrm{dist}(v_i, u)$ **then**
      $\mathrm{L}_i(u) \leftarrow \mathrm{L}_{i-1}(u) \cup \{< v_i, 0, u >\}$
      for $w \in \mathbf{N}(u)$ && visited [w] $\neq 1$ **do**
        if $d[u] + \mathrm{weight}(u, w) < d[w]$ **then**
          $d[w] \leftarrow d[u] + \mathrm{weight}(u, w)$
          $\mathrm{L}_i(u) \leftarrow \mathrm{L}_{i-1}(u) \cup \{< v_i, d[w], \mathrm{u} >\}$
        end if
        if $w \notin PQ$ **then**
          $PQ.\mathrm{insert}(w)$
        end if
      end for
    end if
  end while
end for
return L_n

在此，weight(u, w) 被定义为图中节点 u 与节点 w 之间的距离，它用于量化节点 u 和 w 之间的兼容度。具体地，这一距离值是通过计算节点 u 和 w 在 mashup 数据集中共同出现次数的倒数来确定的。当 weight(u, w) 的值越小，即意味着节点 u 和 w 之间的兼容度越高，反之则表明它们之间的兼容度较低。通过这种方式，我们能够有效地衡量节点间的相互关联程度，为后续的算法设计和优化提供重要的依据。

3.2.2 关键 API 的选取

本文从双重维度对子图的质量进行了深入且细致的剖析。

首先，我们聚焦于算法所推荐的 API 节点集合与用户实际调用的 API 之间的相似度。为了使推荐的 API 更贴近用户的实际需求，我们致力于提升 mashup 结果的命中率，确保子图所包含的 API 节点能够精准覆盖用户实际调用的 API，从而达成高效且

精准的服务匹配。

其次，我们深入探索了子图中所有节点间的距离关系。为了量化这一指标，我们利用历史数据集中 API 的共同出现频次来评估它们之间的兼容性。此方法之所以合理，是因为频繁的共同出现往往意味着这两个 API 之间具有高度的兼容性和协同作用。

在子图权重的度量方法上，我们不局限于仅关注根节点与其他节点兼容性，而是创造性地提出了一种全面考量所有节点间兼容性的新策略。这种策略能够更为准确地捕捉子图的整体结构特征，尤其是其内在的兼容性和协同性，从而为后续的 API 组合推荐提供更为精确且可靠的依据。

综合上述两个方面的考量，我们构建了一个全面而精准的子图质量评估体系。这一体系不仅有助于我们更深入地理解子图的内在特性和优势，同时也为后续的 API 组合推荐奠定了坚实的基础。

鉴于 API 的搜索空间庞大，遍历 API 并计算所有节点间的权重所涉及的时间复杂度极高。为了高效且准确地获取 API，本文采用了基于柱状搜索（beam search）的策略[42]。柱状搜索可视为对贪心算法的优化与拓展，它在每一步选择时，保留前 k 个最佳候选，从而在保证搜索结果的准确性的同时，提升了搜索的效率。

除此之外，当前 Mashup 知识图谱的构建方法主要依赖于计算两节点在 Mashup 数据集中共同出现次数的倒数。虽然这种方法能够在一定程度上强调节点间的兼容度，但却忽视了两节点与其他节点连通性，陷入局部最优解，从而可能限制了子图生成的整体兼容度与准确性。此外，现有的 Mashup 推荐算法往往忽视了 API 根节点的选取方式，而实际上，选择连通性较大的节点作为根节点，更有可能生成兼容度更高的子图，从而提高推荐结果的质量和准确性。

因此，为了弥补这些不足，本文提出了两个面向 Mashup 推荐的知识图谱权重优化策略和根节点选择优化策略。这些策略旨在进一步提升知识图谱的构建质量，优化根节点的选择方式，从而提高推荐算法的性能和准确性。具体的 API 选择算法实现如算法 3 所示。

算法 3：关键 API 选择算法

输入：候选功能 API 集合$Categories = \{C_1, ..., C_n\}$, api间最短距离标签集合Ln.

输出：排名最高的一个子图.

1.C = $\{C_1, ..., C_n\}$

2.Sort C by $\overline{\text{degree}}$,按每个功能 API 集合的平均度数值从高到低排序并重新编号

3.$Beam \leftarrow Top(k, \text{api} \in C_1, \text{degree})$, 取 C_1集合中前 k 个度数值最高的 API

4.**for** $i \leftarrow 2$ **to** n **do**

5.　newBeam = $\{\oslash\}$　建立一个新的空 Beam

6. Map (newV, cos t) $= \{\oslash\}$ 建立 Beam 向 cos t 的空映射
7. **for each** $V \in$ Beam **do**
8. **for each** $api_c \in C_i$ **do**
9. **if** $\forall api_v \in V \left(L\left(api_v\right) \cap L\left(api_c\right) \neq \oslash\right)$ **do**
10. $delta \leftarrow \sum_{api_v \in V} \frac{dist(api_c, api_v)}{degree(api_c) * degree(api_v)}$
11. $newV \leftarrow V \cup \{api_c\}$
12. $cost \leftarrow cost + delta$,增加该结点后子图的总的代价
13. Add(newV, newBeam), 将结果加入到 newBeam 中
14. Add ({newV, cos t} , Map) , 建立 newV 与 cos t 的映射
15. **end if**
16. **end for**
17. **end for**
18. sort NewBeam by cost, 按cost 从低到高排序
19. $Beam \leftarrow Top(k$, newV $\in newBeam$, cos t), 取$newBean$前 k 个代价最小的V 组成新的$Beam$
20. **end for**
21. **return** $Top(1, V \in Beam$, cos t)

针对输入的候选功能 API 集合，本文首先计算每个集合的平均度数，并按照该值从高到低的顺序进行排序。随后，本文引入根节点选择优化策略选择平均度数值最高的集合作为起始集合，旨在增加根节点与其他节点之间的连接可能性，进而提升子图的兼容性。通过这一策略，本文能够更有效地从候选 API 中筛选出关键节点，为后续构建高质量子图奠定坚实基础。接着，从该集合中选取度数排名前 k 位的节点作为起始节点，其中参数 k 代表柱状的大小。

在算法的第 10 行至第 14 行中，为了精确计算加入当前节点后所带来的代价变化，本文首先采用算法 1 来确定节点之间的距离。随后，为了提升选择高连接度节点的概率，并增强节点与子图中其他节点的连接可能性，本文引入知识图谱权重优化策略。具体而言，我们将节点间的权重优化为节点共同出现频次的倒数除以两个节点度的乘积，通过这样的处理，能够更有效地衡量节点间的相对重要性，并倾向于选择那些具有高连接度的节点。这一优化策略不仅有助于构建一个更加紧密的子图结构，还能更准确地评估每次添加节点对子图构建的综合影响。一旦代价计算完成，算法将生成一个新的子图，并将其加入到候选集合中，以供后续的比较和选择。通过这样的处理方式，我们能够更精准地控制子图的生成过程，从而为用户推荐更加优质的 API。

然后，在算法的第 18 行和第 19 行，本文依然保留候选列表中排序靠前的 k 个结

果，以确保搜索的高效性最后，返回排名最高的子图作为算法的输出结果。

3.2.3　子图的扩展生成

本文利用 HL 算法计算距离，成功简化了计算过程并提升了运算速度。然而，需要特别指出的是，在节点选择环节，我们并未将实际存在的边纳入考量，导致当前成果仅限于获取子图的顶点集合。为了向用户提供更加兼容的 API 组合，我们必须将这些孤立的节点相互连接，形成一个连通的子图。因此，我们接下来的工作重心将是从已选定的顶点集出发，通过逐步扩展，构建出一个完整且连通的子图结构。这将有助于确保 API 之间的顺畅协作，实现系统的高效运行。具体如算法 4 所示。

算法 4：子图扩展生成算法

输入：关键 API 节点集合 V，api 间最短距离标签集合 L

输出：最小斯坦纳树的路径点集合

```
E ← ⟨∅⟩
E ← prim(V)  输入API节点并利用Prim算法找出子图的关键边
for each ⟨s, t⟩ ∈ E do
    dist ← +∞
    for each h ∈ L(s) ∩ L(t) do 找两节点间最短距离所经过的中间节点
        if dist > dist(s, h) + dist(t, h)  do
            mid ← h
            dist = dist(s, h) + dist(t, h)
        end if
    end for
    while s.par ≠ mid do 如果当前节点不为中间节点则遍历节点前驱
        s = s.par
        V ← V ∪ ⟨s⟩ 将当前节点至中间节点最短路径上的节点放入 V
    end while
    while t.par ≠ mid do
        t = t.par
        V ← V ∪ ⟨s⟩
    end while
end for
return sorted(V) 将节点集合按照节点的度从高到低进行排序并返回
```

针对输入的关键 API 节点集合 V，首先，我们通过 prim 算法获得了子图的关键边集合 E。接着，对集合 E 中的每条边进行遍历，并将最短距离 dist 重置为正无穷大，以确保初始状态。随后，我们聚焦于寻找每对关键点 s 和 t 的标签集合 L(s) 与 L(t) 的交集，并在其中确定两节点间最短路径上的中间节点 h。一旦找到符合条件的中间节点 h，立即更新 mid 为 h，并计算 s 到 h 与 t 到 h 的距离之和，更新 dist 值。之后，若当前节点并非中间节点，则进一步遍历其前驱或后继节点，将当前节点至中间节点最短路径上的关键节点补充至 V 中。最终，返回构建完成的路径点集合 V 作为算法输出，该集合包含了构建最短路径所需的关键节点。

最终，为了显著增强 API 推荐结果与用户需求之间的契合度，我们引入了一种基于节点度的先进排序算法。该算法精妙地依据节点的度（即其在网络中的连通性）对推荐结果进行降序排列，确保那些与用户需求高度相关的节点能够优先展示给用户。这种创新的排序策略不仅大幅提升了推荐结果的相关性，还显著优化了用户体验，使用户能够更迅速、更直接地找到满足其特定需求的 API。

第四章　实验与实例

基于上述方法论，本文设计并实现了一种新颖的基于静态中心标注法的 mashup 组合推荐算法 FAR。为了验证所提出方法的有效性与优越性，本文设计并实施了一系列实验，旨在解答以下两个核心研究问题：

研究问题一：本文所提出的方法是否能够高效且精确地推荐用户所需的 API?

为了深入探究此问题，本文首先利用静态中心标注法构建了图的离线索引结构。随后，我们采用经过优化后的子图评价标准进行 Mashup 子图的生成，并巧妙地运用柱状搜索策略提升子图的生成效率。这一方法的核心目标在于有效解决传统 Mashup 推荐结果与实际使用场景脱节，以及子图搜索空间过于庞大等难题，进而大幅度提升推荐算法的精准度与运行效率。为了验证这一方法的有效性，我们选取了 Mashup 数据集进行详尽的实验验证。并将所得结果与前沿的 API 推荐方法进行了深入细致的对比分析。

研究问题二：根节点优化策略与子图权重优化策略在 mashup 推荐领域是否具有泛化性?

在深入探究研究问题一的基础上，为了验证优化策略的泛化性，我们将根节点优化策略与子图权重优化策略应用于现有的子图搜索算法中，通过实证分析来检验其在 Mashup 推荐领域的通用性。这一验证工作不仅有助于深入理解优化策略的作用机制，还能为 Mashup 领域的知识图谱生成与子图搜索提供有益的启示，推动相关技术的创新与发展。

4.1　实验设计

本文选取了从 ProgrammableWeb 网站中爬取的 mashup 数据集作为实验对象. 以下是实验设计的详细阐述：

4.1.1　数据集描述

本研究所采用的数据集源自 ProgrammableWeb 的历史记录，涵盖了丰富的 APP 及其所使用的 API 信息。具体而言，该数据集包括 APP 的名称、实现的功能、所使用的 API，以及 API 的名称和功能等关键信息。此 mashup 数据集共计包含 4870 个 APP 记录，与之相对应的 API 数据集则包含 1233 个 API 记录。mashup 与 APIs 数据集示例如下表所示：

表 4.1 mashup 数据集示例

APP 名称	相关 APIs	功能
Json API App	Twitter/Analytics SEO	Application Development/ Products
WaifuAI Android	WaifuAI	Artificial Intelligence/ Chat/Machine Learning
Page Experience Checker	Google PageSpeed Insights	SEO/Analytics/ Marketing/Search
Lot Near You APP	Google Maps/Google Chart/ eBay Product Services/Google Geocoding	SEO/Analytics/ Marketing/Search
Music on Tube	Flickr/YouTube/Last.fm	Music/Photos/Video

表 4.2 APIs 数据集示例

API 名称	API 功能
YouTube	Video/Media
Google	Maps Mapping/Viewer
Twitter	Social/Blogging

通过这一全面的数据集，本文旨在深入探究 mashup 组合推荐算法的有效性和性能，以期为用户提供更为精准、高效的 API 推荐服务. 在实验过程中，本文将充分利用数据集中的各项信息，确保实验结果的客观性和准确性。

4.1.2 评估方法

为了全面评估本文所提出方法的实际效果，本文首先对柱状搜索中使用不同的 k 值进行实验。实验中，本文以命中率和运行时间作为评判标准，细致地选取并测试了多个 k 值，旨在找到最优的 k 值设置，以确保推荐算法在准确性和效率上达到最佳平衡。

鉴于目前 Web API 的推荐领域尚缺乏通用的基准，我们进一步采用了一些具有代表性的方法作为基线，以此验证我们提出的方法的有效性。

1）Random：该方法从功能所对应的 API 集合中随机选择一组共同覆盖查询关键词的节点，然后找到最小生成树将所选节点与所有生成树中节点数最少的节点连接起来

2）Greedy：该方法从功能所对应的 API 集合中随机选择一组共同覆盖查询关键字的节点。然后，它将这些节点作为初始根节点，并不断生长树，直到所选节点互连。当树生长时，应用贪婪启发式，以便首先选择包含最多查询关键字的邻居。

3）KeyKG[41]：它是一种针对图结构数据的关键词搜索算法，它基于群斯坦纳树实现语义搜索，并借助高效的近似算法和 HL 算法结构，在毫秒时间内为大型知识图谱提供可靠且质量上乘的搜索结果。

4）MTFM[22]：它是一个基于多模型融合和多任务学习的神经网络框架。MTFM 使用卷积神经网络生成需求表示，并引入功能交互组件来对 mashup 和 Web API 之间的功能交互进行建模，以预测候选 API。为了确保 MTFM 模型在精准度与实用性之间达到平衡，本研究仅选取了模型推荐结果的前十位进行性能评估和比较。

此外，为了进一步验证优化策略的有效性，本文将根节点优化和子图权重优化策略应用于 KeyKG 算法中。通过对比实验，本文细致地比较了这些优化策略在 KeyKG 算法中的表现，并对其效果进行了全面的分析和讨论。

4.1.3 评估指标

为了全面且精准地评估 APIs 推荐结果的效果，本文引入了平均运行时间、推荐节点数、平均精确度（MP）、平均召回率（MR）、归一化折损累计增益（NDCG）指标作为度量标准。

1）平均运行时间：为了精确评估算法在运行效率方面的性能，本研究将算法在处理应用开发者不同数量的关键词查询时所需的平均处理时间作为核心评价指标。该指标不仅直观地反映了算法在处理大规模数据集时的效能，更为算法的优化和深入改进提供了坚实的数据基础。

在确保 API 间兼容性达到较高水准的前提下，我们特别强调运行时间的重要性，因为用户的核心需求在于迅速获取满足软件功能需求的 API 组合推荐结果，而非仅仅使用与其他应用相似的 API。因此，通过这一指标的考量，我们能够更好地满足用户的实际需求，提升算法的实际应用价值。

2）推荐节点数：指标能够反映出推荐系统在处理过程中所涉及的 API 节点数量。当推荐系统返回的节点数量较少时，这通常意味着系统能够更专注、更精准地为用户推荐符合其需求的 API。这样做不仅有助于减少用户在选择过程中的困惑和犹豫，还能使用户更加迅速地找到最符合其需求的选项。

3）平均精确度（MP）：MP 指标是用于全面衡量算法在推荐 API 组合时整体性能的关键标准。具体而言，它计算的是模型推荐的 API 组合（RLi）与 APP 实际使用的 API（RLapp）之间的平均精度。这一指标直接反映了推荐算法在为用户提供推荐时，其推荐内容与用户实际需求之间的吻合程度。通过 MP 指标的衡量，我们能够更准确地了解推荐算法在为用户提供 API 组合时的表现，并据此对算法进行有针对性的优化，以提高推荐的精准度和用户满意度。

鉴于不同数据集会对结果产生的显著影响，为确保算法性能评估的准确性和客观性，避免随机选择少量数据集可能带来的偏差，同时验证本算法的运行效率，本文特选用全面的 mashup 数据集进行深入的算法验证，并据此计算 MP。

$$MP = \frac{1}{K}\sum_{i=1}^{K}\frac{\left|RL_i\right| \cap \left|RL_{app}\right|}{\left|RL_i\right|}$$

4）平均召回率（MR）：MR 是衡量推荐算法在多个 mashup 服务组合上整体性能的指标，通过计算召回率的平均值来评估算法在多个服务组合上的综合表现。与 MP 不同，MR 侧重于评估算法在覆盖用户实际需求上的能力。采用 MR 作为评估标准，能确保结果的客观性和准确性，为算法优化提供依据。此外，为了保障评估结果的全面性和客观性，我们在完整的 mashup 数据集上进行了详尽的实验，从而确保所得结论的可靠性。

$$MR = \frac{1}{K}\sum_{i=1}^{K}\frac{\left|RL_i\right| \cap \left|RL_{app}\right|}{\left|RL_{app}\right|}$$

5）归一化折损累计增益（NDCG）：是一种专为评估推荐结果排序质量而设计的指标。它综合考虑了 API 推荐结果与用户需求之间的相关性以及这些结果在排序列表中的位置，从而全面、准确地评估排序算法的性能。$rel_i = 1/0$ 表示第 i 个 Web API 是否真正与当前的 mashup 相关。DCG 从结果列表的顶部累积到底部，较低排名的每个结果的增益会进行折损。

$$DCG = rel_1 + \sum_{i=2}^{N}\frac{rel_i}{\log_2(i+1)},$$

$$IDCG = \sum_{i=1}^{|RL_{app}|}\frac{2^{rel_i}}{\log_2(i+1)},$$

$$NDCG = \frac{DCG}{IDCG}.$$

综上所述，通过采用平均运行时间、推荐节点数、MR、MP、NDCG 作为评估指标，本文能够全面、客观地评估 API 组合推荐结果的准确性和算法的运行效率，为后续的研究和应用提供坚实的基础。

4.2 实验结果

4.2.1 柱状搜索 k 值选取

针对 mashup 数据集，本文深入探讨了不同 k 值对柱状搜索算法效果的影响。经过细致的实验对比和分析，我们观察到 k 值的变化对精准度和运行时间均产生了显著影响。具体实验结果如下表所示，当 k 值从 1 增加到 5 时，精准度呈现先上升后下降的趋势，而运行时间则持续增加。

值得注意的是，实验环境对算法的性能表现也有一定影响。为了确保实验的公正性和可重复性，本文在 Windows 10 系统、i7-1065G7 CPU、16GB 内存以及 Java 1.8 版本的环境下进行了所有实验. 这样的实验配置能够为我们提供一个稳定且可靠的性能评估基础。

表 4.3　不同 K 值实验结果

K 值选取	精准度 (%)	整体运行时间（秒）
1	29.49	0.782
2	30.36	1.010
3	28.08	1.249
4	28.00	1.608
5	27.22	1.918

为了平衡精准度和运行时间，本文选择 k 值为 2 进行下一步实验。这一选择基于以下考虑：相较于 k=1，k=2 时的命中率略有提升，虽然运行时间有所增加，但仍处于可接受范围内；而当 k 值继续增大时，精准度开始下降，且运行时间的增长也更为明显。因此，本文认为 k=2 是一个较为合理的折中选择，能够在保证一定精准度的同时，维持相对较高的运行效率。

这一结论为我们后续的实验和研究提供了重要参考，有助于进一步优化柱状搜索算法在 mashup 数据集上的应用效果。

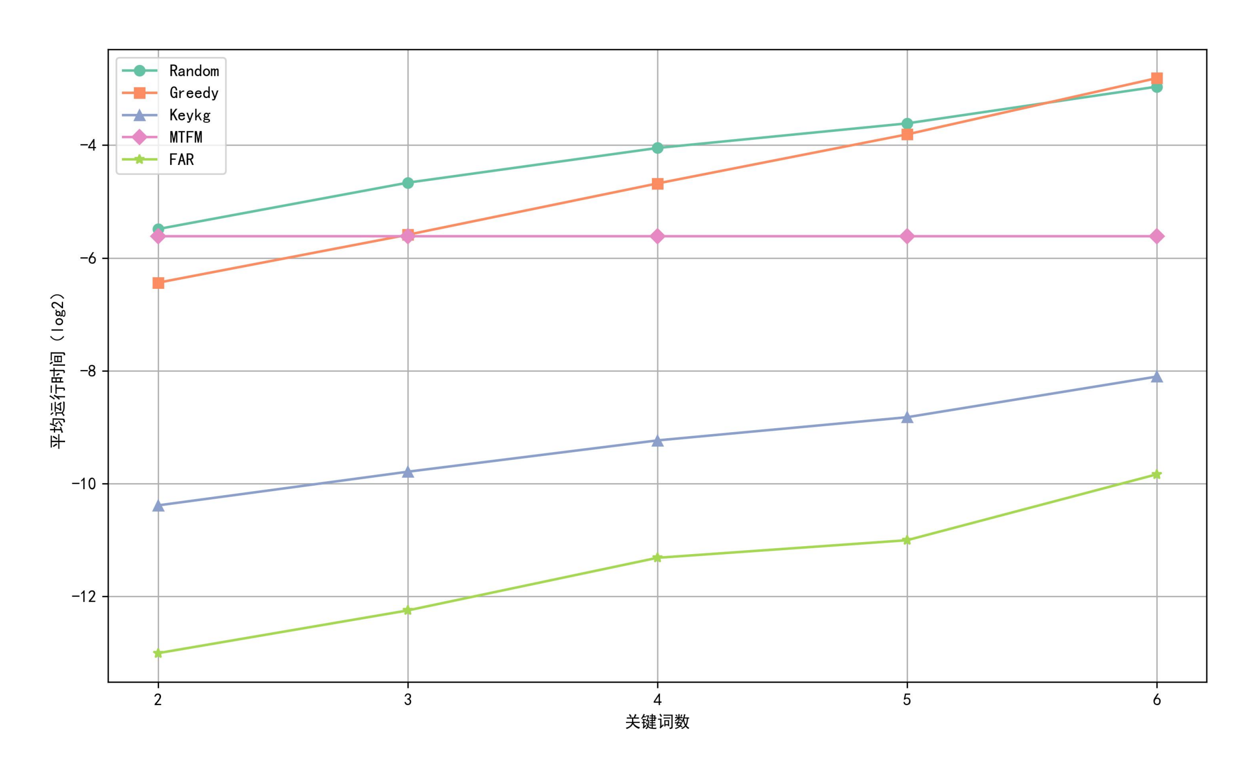


图 4.1　平均运行时间

4.2.2　平均运行时间比较

接下来，我们针对五种不同算法的效率进行了详尽的比较，通过衡量它们在处理应用开发者不同数量的关键词查询时，找到合适 API 所需的计算时间来进行评估，实验结果如图 4.1 所示。

实验结果表明，FAR 算法在响应不同数量的关键词查询以找到合适 API 时，展现出显著的计算时间优势。从平均搜索时间的对数值（以 2 为底）来看，随着关键词个数的递增，FAR 算法的运行时间依然保持在较低水平，明显优于 Random、Greedy、KeyKG 和 MTFM 等算法。

具体而言，当关键词个数为 2 时，FAR 算法的平均搜索时间对数值约为-13，这远低于其他算法。即便在关键词数量增加至 6 个时，FAR 算法的平均搜索时间对数值虽有上升，但仍保持在-10 的较低水平。相比之下，Random 和 Greedy 算法在关键词数量增加时，搜索时间显著增长。

在实验中，MTFM 算法作为一种神经网络方法，其搜索时间虽相对稳定，但在性能上却明显逊色于 FAR 算法。与此同时，尽管 KeyKG 算法亦展现出不俗的性能，但在各类关键词数量的测试中，FAR 算法均展现出更低的搜索时间，这充分证明了 FAR 算法在处理不同数量关键词查询时具有更高的效率。

综上所述，FAR 算法在时间效率上占据了显著优势。不论是在处理少量还是大规

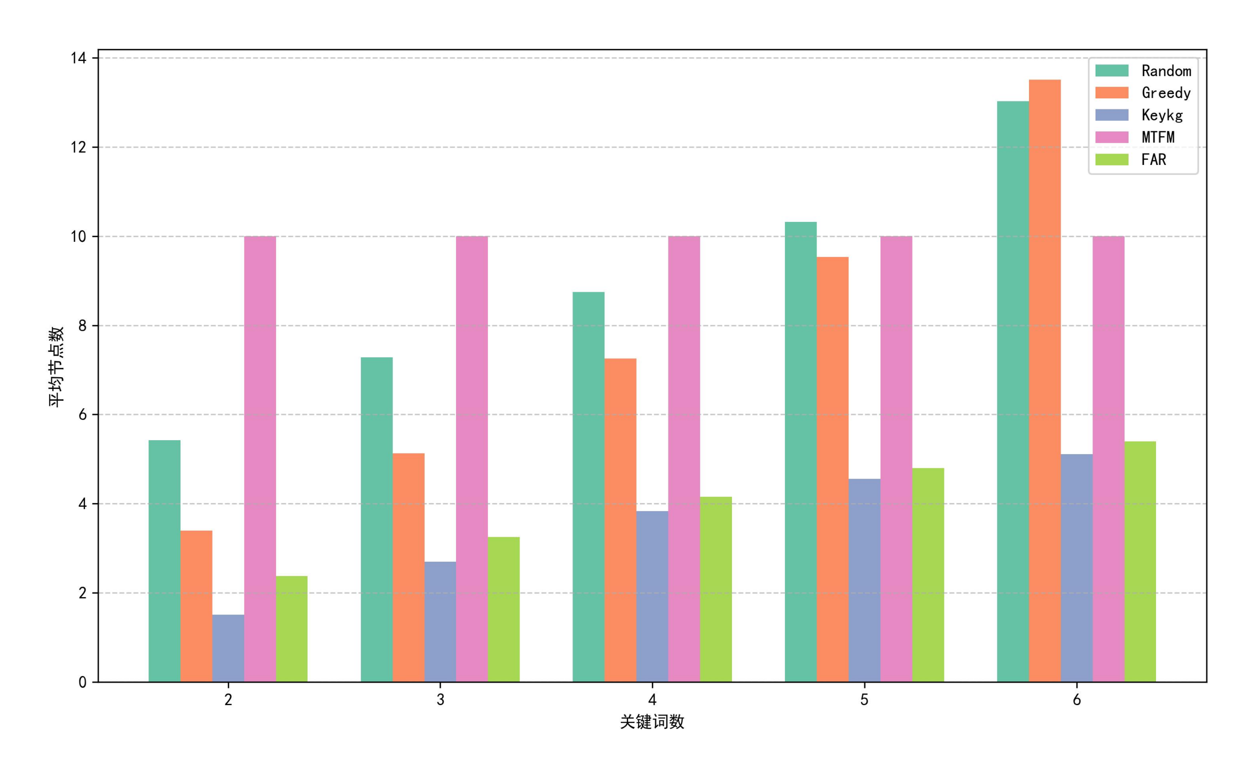


图 4.2　推荐节点数

模的关键词查询时，FAR 算法均能迅速而准确地找到解决方案，这一特性对于提升应用开发的效率以及优化用户体验而言，具有不可忽视的实际价值。

4.2.3　推荐节点数比较

针对五种不同算法在推荐结果数量方面的性能，我们进行了详尽且系统的比较分析。在评估这些算法处理不同数量的关键词查询时，我们观察到了以下显著的规律，实验结果如图 4.2 所示。

首先，根据实验数据显示，随着关键词数量的增加，所有算法所需的平均节点数都呈上升趋势。然而，在这一普遍趋势中，FAR 算法凭借其稳定的低节点数表现突出。具体而言，当关键词数量从 2 增加到 6 时，FAR 算法的平均节点数仅从约 2 个增加到约 5 个，展现出相对较小的增长幅度。

相比之下，Random 算法和 Greedy 算法在关键词数量较少时节点数相对较低，但随着关键词数量逐渐增加，它们的节点数增长趋势更为显著。特别是当关键词数量达到 6 时，这两种算法的平均节点数已经超过了 FAR 算法的两倍，这显示了 FAR 算法在关键词数量增加时的高效性和经济性。

由于 MTFM 算法在推荐 API 之前需要预设生成 API 的数量，因此，无论关键词数量如何变化，其节点数均保持恒定。这一特性表明 MTFM 算法在处理不同数量的关键词时可能缺乏必要的灵活性。相比之下，FAR 算法能够根据关键词数量的动态变化自

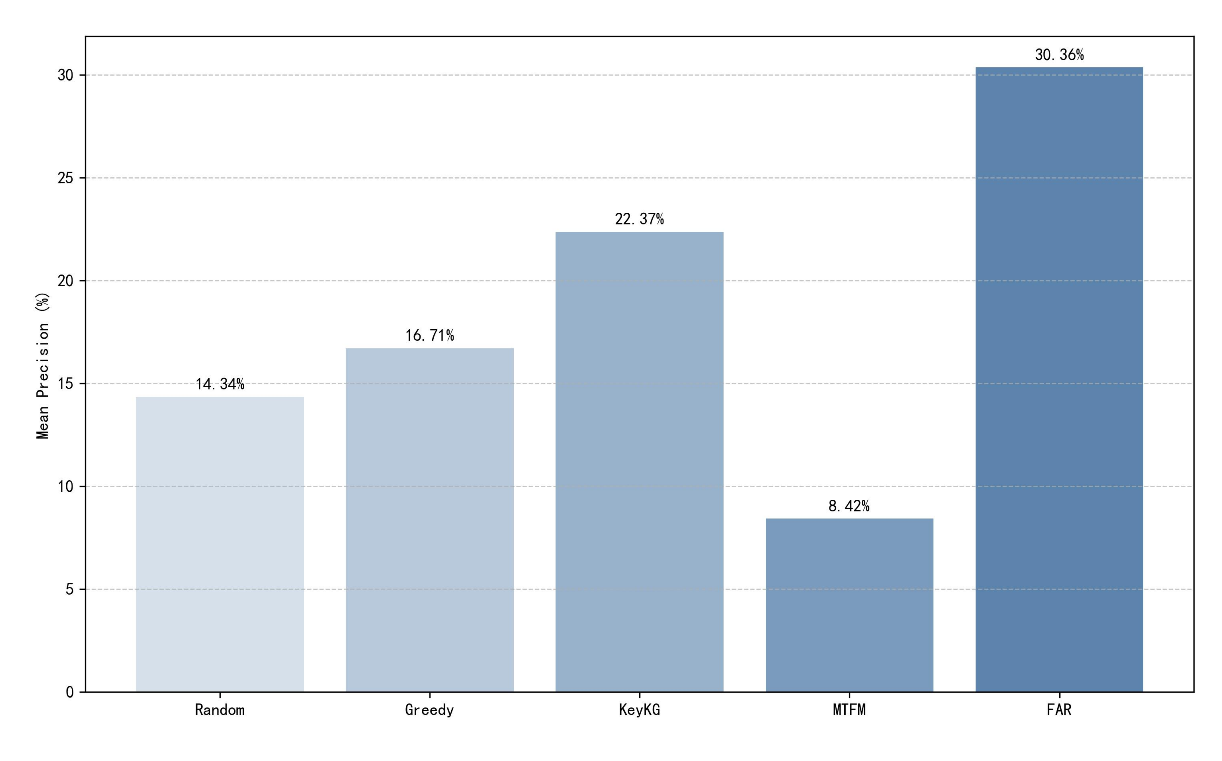


图 4.3　平均精确度

适应地调整所需节点数，从而在保持较低节点数的同时，实现更好的推荐效果。

KeyKG 算法在各个关键词数量下展现出了最低的推荐 API 个数。尽管 KeyKG 算法在关键词数量较少时表现出色，但随着关键词数量的增加，其节点数增长相较于 FAR 算法呈现出明显的上升趋势。

综上所述，FAR 算法在关键词数量增加时能够用较少的节点获取更好的结果，彰显了其在节点利用效率方面的优势。因此，在处理涉及大量关键词的任务时，FAR 算法是一个更为高效和经济的选择。

4.2.4　平均精确度比较

接着，我们对五种不同算法在平均精确度指标上的性能进行了比较分析。鉴于平均精确度的计算过程中，数据集的选择对算法结果有着显著的影响，为确保实验结果的客观性和准确性，我们采取了全面的数据集覆盖策略，即选取了所有可用数据集进行严格的实验验证。通过这一方法，我们有效地避免了因随机选取实验数据集而可能引入的偏差。实验结果如图 4.3 所示。

其中，Random 算法因其内在的随机性，导致推荐的精确度相对较低。具体而言，其 MP 值仅为 14.34%，表明在随机选择推荐结果时，正确推荐的 API 数量较少，效果不尽人意。

Greedy 算法相较于 Random 算法，在一定程度上进行了优化。它通过贪心策略尝试

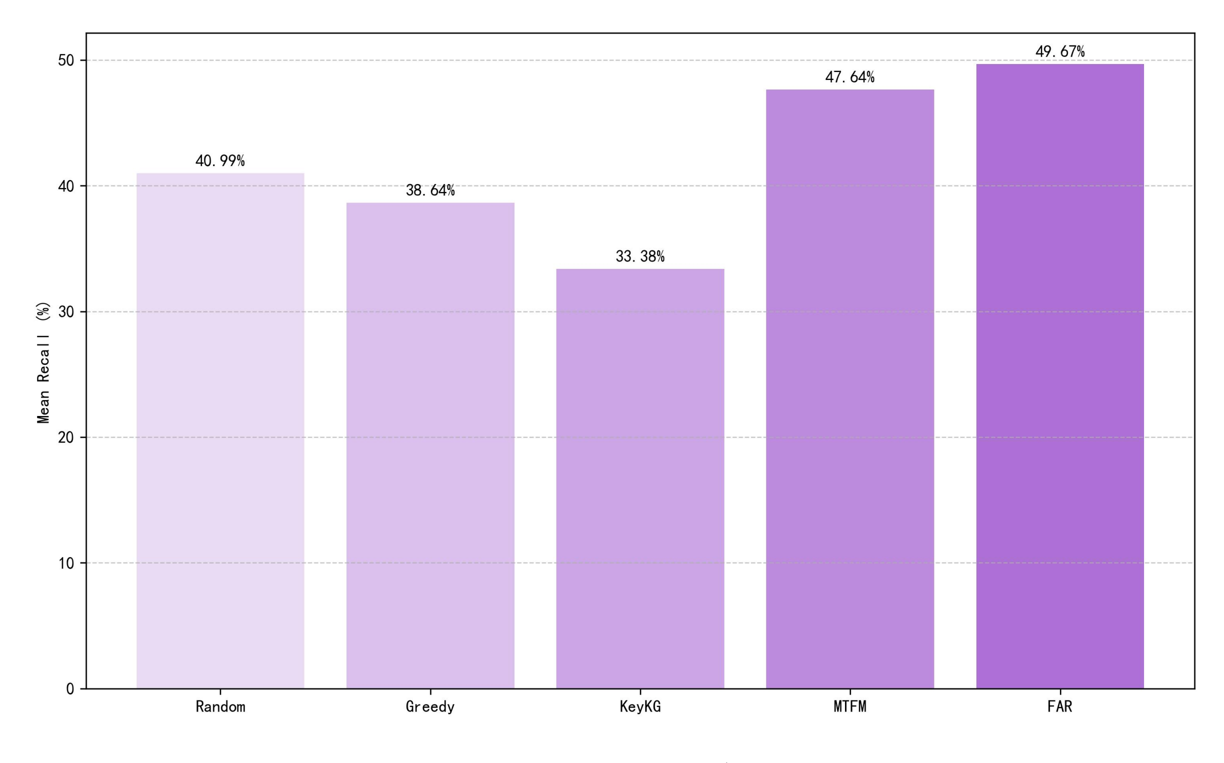


图 4.4　平均召回率

选择当前最优的解,但这种策略受限于局部最优解的约束,使得其 MP 值提升至 16.71%,虽然有所提升，但提升幅度相对有限。

然而，MTFM 算法在应用中遇到了一个显著的问题：由于其采用每次固定推荐 10 个 API 的策略，而实际应用场景中所需 API 的数量往往少于此数，这导致了正确推荐的 API 数量在总推荐数中占比偏低。因此，MTFM 算法的仅为 8.42%，在五种算法中表现最低。

KeyKG 算法通过引入 Hub Label 的概念，能够高效地进行全局最小斯坦纳树的搜索，有效提升了推荐的准确性，其平均精确度达到了 22.37%，显示出其在推荐系统领域的良好性能。

在众多算法中，FAR 算法的表现尤为突出。该算法凭借其独特的子图评价方法和高效的搜索优化策略，显著提升了推荐的精确度。实验结果显示，FAR 算法的平均精确度高达 30.36%，远超其他算法。

4.2.5　平均召回率比较

随后，我们进一步对五种不同算法在平均召回率指标下的性能进行了细致的分析。为确保评估结果的客观性和可靠性，我们依然沿用了全面的数据集覆盖策略，对所有可用的数据集进行了详尽的实验验证。这些实验结果通过图 4.4 进行了直观的展示，为我们提供了各算法在召回率方面的具体表现。

在平均召回率指标的对比中，我们发现 Random 算法的召回率达到了 40.99%，这在一定程度上体现了其随机性的优势，能够在一定程度上覆盖到较多的相关项目。然而，Greedy 算法的召回率略低于 Random 算法，为 38.64%，这可能是由于其贪心策略在追求局部最优时牺牲了部分全局召回率。

值得注意的是，MTFM 算法在平均召回率上取得了 47.64% 的显著成绩。这一成果主要归因于其每次固定推荐 10 个 API 的策略。尽管这一策略可能在一定程度上牺牲了推荐的精确度，因为它不总是针对用户需求精准地推荐 API，但这种策略确保了 MTFM 算法在广泛的 API 推荐中能够覆盖更多相关的选项，从而增加了用户找到所需 API 的可能性。

KeyKG 算法在平均召回率方面的表现稍显不足，仅达到了 33.38%。这一结果主要归因于其采用的近似最小斯坦纳树策略。尽管该策略确保了节点间权重的最小化，但所选节点可能并不完全符合用户的实际需求，从而影响了算法的召回率。

在所有评估的算法中，FAR 算法再次凸显了其卓越性能。其平均召回率高达 49.67%，显著超越了其他算法。这一结果不仅证明了 FAR 算法在推荐系统领域中的高精度特性，同时也展现了其在召回率方面的出色能力。

4.2.6 归一化折扣累积增益比较

最后，我们再次采用全面的数据集覆盖策略，对五种不同算法在 NDCG 指标下的性能进行了详尽的分析。实验结果如图 4.5 所示。

在 NDCG 的对比中，Random 算法取得了 29.78% 的得分，这揭示了其随机排序的方式在捕捉推荐结果与用户偏好相关性方面的局限性。然而，Greedy 算法在 NDCG 指标上有所提升，达到了 34.36%，这主要得益于其贪心策略，该策略在每次选择时都倾向于选取当前最优的节点，从而在一定程度上增强了推荐结果与用户偏好的相关性。

尽管 KeyKG 算法在 NDCG 上取得了 36.51% 的得分，较之前两种算法有所提升，但这一结果仍受到一定限制。KeyKG 算法在搜索策略上的优势使得推荐结果的精确度得以提升，然而，由于缺乏有效的排序策略，导致其在 NDCG 这一同时考虑相关性和排序质量的指标上表现相对较低。

值得注意的是，MTFM 算法在 NDCG 评估中取得了 56.33% 的显著成绩。这一优异表现主要归功于其独特的多任务学习与特征学习策略。MTFM 算法通过整合不同任务的信息和特征，对推荐结果进行排序，确保它们按照与关键词的相关性进行排列输出。因此，在 NDCG 这一强调排序位置与项目相关性的指标上，MTFM 算法有效地提升了排序结果与用户偏好的匹配度。

在所有算法中，FAR 算法再次凭借其卓越的性能脱颖而出。FAR 算法考虑了节点

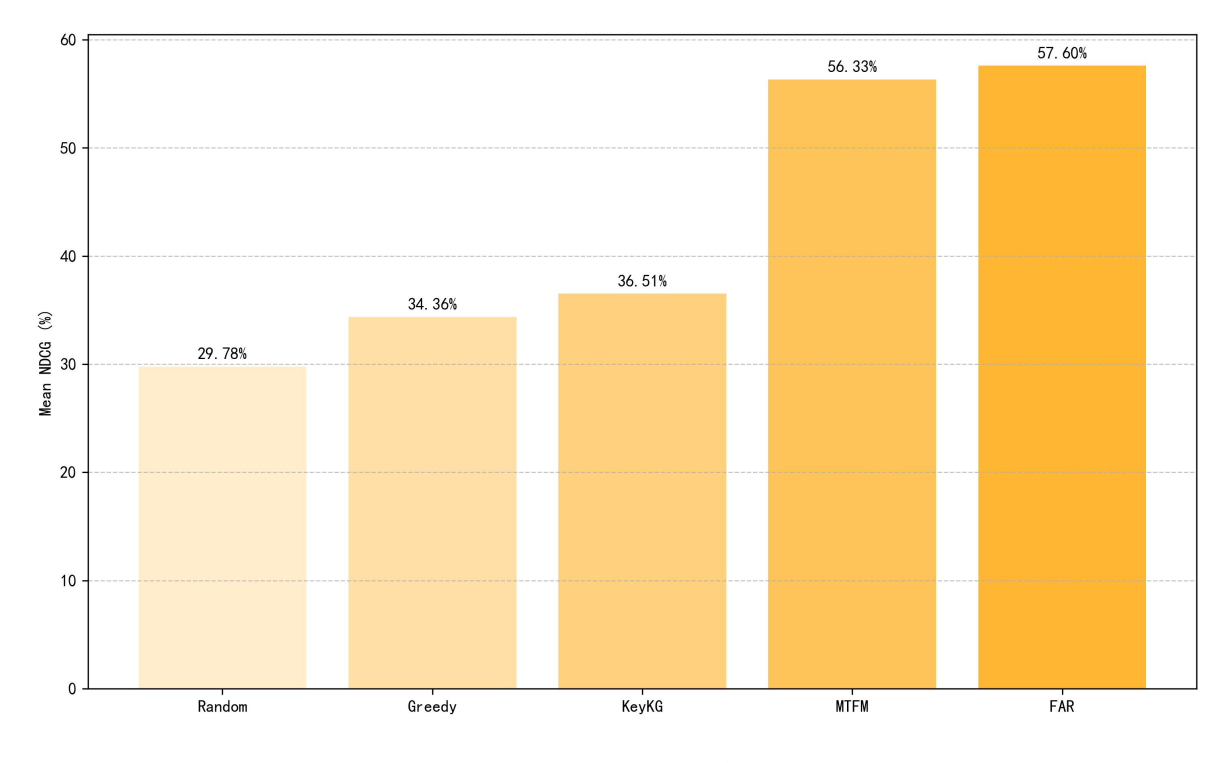


图 4.5　NDCG 比较

之间的连通性，通过高效的策略对每个节点的连通性进行评估和排序，并在输出推荐时优先推荐那些重要性更高的节点。这一策略使得 FAR 算法在 NDCG 评估中取得了高达 57.60% 的得分，获得了最优的结果，证明了其在推荐系统领域的出色表现。

这一结果不仅证实了 FAR 算法在推荐系统领域中的高精确度，也凸显了其在排序结果相关性方面的优异性能。这些显著的性能提升不仅为 FAR 算法在推荐系统领域的应用提供了强有力的支撑，也为后续推荐算法的研究与开发提供了宝贵的参考和启示。

4.2.7　实验结果分析

实验对五种 API 推荐算法进行了详尽的性能对比，通过量化评估这些算法在运行时间、推荐节点数、MP、MR 以及 NDCG 等多维度指标上的表现。实验结果显示，FAR 算法在各项指标上均展现出显著优势，这主要得益于其独特的策略组合，包括静态中心标注、增强子图评估以及创新的优化策略，这些策略有效整合了网络兼容性和节点重要性信息。

在运算效率方面，FAR 算法利用了中心标注法，显著优化了寻找两节点间最短路径的时间成本。与 KeyKG 算法相比，FAR 通过柱状搜索策略更有效地缩减了冗余的搜索空间，在应对大量 API 推荐请求时展现出了卓越的效率。相较之下，MTFM 算法作为深度学习算法，由于其计算复杂度较高，导致处理任务时所需时间相对较长；而 Random 和 Greedy 算法作为传统的动态规划方法，在处理大规模搜索空间时，由于子

树扩展带来的计算负担较重，因此其运行效率较低。

在推荐节点数方面，FAR 算法以提高命中率为核心目标，它倾向于搜索连通性高、实用性更强的节点，而非单纯追求整体兼容性最高的子树结构。因此，相较于 KeyKG 算法，FAR 算法的推荐节点数有所增加，以更好地满足实际应用中的需求。MTFM 算法作为一种深度学习算法，在推荐节点数方面需要人为规定输出的数量，因此其结果数量相对稳定，受算法自身特性影响较小。然而，Random 和 Greedy 算法在推荐节点数方面则面临挑战。由于初始节点选取策略的不佳，这两种算法往往难以选取到高兼容性的初始点，这导致在后续的扩展过程中，生成的子树节点数量过多，既增加了计算负担，也可能降低了推荐结果的准确性和实用性。

在召回率与准确率方面，FAR 算法充分利用了网络兼容性和节点连通性的信息。它考虑的是所有节点间的权重，而非仅限于根节点到其他节点的权重。这种策略使得所选子图的节点更为紧密，整体兼容性更高。同时，FAR 算法倾向于选择度数更大的节点，即那些与其他 API 兼容度更高且使用频率更高的节点。此外，通过柱状搜索算法，FAR 算法能够精准地筛选出前 k 个最优结果，有效避免了过分强调 API 间兼容性而忽视命中率的倾向。这些策略确保了 FAR 算法所推荐的 API 不仅与用户需求高度相关，而且在实际应用中具有较高的实用性和通用性。

相较而言，KeyKG 算法在构建子图时更注重根节点与子图其他节点的距离，并倾向于用较少的 API 来覆盖 APP 的功能以追求整体最高的兼容性。然而，这种策略可能忽略了 API 在实际应用中的使用频率和重要性，从而导致推荐的 API 与用户需求不完全匹配。MTFM 算法作为一种基于节点特征的推荐算法，在推荐过程中忽略了网络的兼容性信息。该算法仅依赖节点特征进行推荐，并倾向于推荐大量节点，导致其具有较高的准确率但召回率较低。相比之下，Random 和 Greedy 算法在功能集合所依赖的 API 选择上存在随机抽取的情况，缺乏对网络兼容性的系统考量。这种随机性导致其在推荐的精确性方面存在不足，难以提供满足特定需求的精准推荐结果。

在 NDCG 方面，FAR 算法表现出色。这是因为它整合了节点信息，并根据节点连通性对输出结果进行排序，从而生成高质量的推荐。类似地，MTFM 算法也取得了较高的 NDCG 分数，这得益于其利用 API 特征与用户需求之间相似性的度量方式。相比之下，KeyKG、Random 和 Greedy 算法缺乏针对 NDCG 指标优化的策略，因此表现不佳。

4.2.8 优化策略泛化性分析

为了探究根节点优化策略和子图权重优化策略在 Mashup 推荐领域的泛化能力，我们基于完整数据集对 KeyKG 算法进行了扩展，并引入上述两种优化策略。通过一系列

性能指标的评估，验证策略的有效性，结果如下表所示。

表 4.4　KeyKG 模型与策略实验结果

模型	运行时间	节点数	MP(%)	MR(%)	NDCG
KeyKG 原算法	5.61 秒	2.60	22.37	33.38	0.36
根节点优化	5.36 秒	2.63	22.61	34.17	0.37
子图权重优化	5.96 秒	2.66	23.20	35.47	0.38
根节点优化 + 子图权重优化	6.99 秒	2.70	23.20	35.96	0.39

首先，我们评估了根节点优化策略对 KeyKG 算法的影响。实验结果表明，实施根节点优化策略后，算法的整体运行时间有所减少。同时，我们也注意到平均节点数略有增加，这表明优化后的算法在构建子图时能够覆盖更多相关联的节点。在性能方面，MP、MR 和 NDCG 等关键指标均有所提升，这进一步证明了根节点优化策略的有效性。

根节点优化策略的核心在于，它在进行搜索之前会根据节点的平均度数对功能 API 集合进行细致的排序，优先选择度数高的节点作为子图的根节点。这种策略的优势在于，它不仅能增强根节点与后续加入节点的兼容性，还能确保搜索过程从一开始就聚焦于那些高度相关且被广泛使用的 API，从而有效提升搜索的命中率。

随后，我们分析了子图权重优化策略对 KeyKG 算法的影响。实验结果显示，虽然应用该策略后算法的整体运行时间略有增加，但鉴于其在性能上的显著提升，这一时间成本是完全值得的。同样地，平均节点数的增加进一步验证了优化策略在拓展搜索范围方面的作用。在性能方面，MP、MR 和 NDCG 指标均实现了显著的提升。

在计算两节点间距离时，通过除以两点度数的乘积，这一策略能够显著提升子图整体的兼容性。它有效缓解了某些 API 间距离较近但却难以与其他 API 联通的问题，确保了搜索过程中能够更全面地考虑到 API 之间的相互关系和整体网络结构。通过这种优化方式，子图权重策略不仅增强了搜索的广度和深度，还进一步提高了算法在识别高质量 API 组合方面的能力，从而显著提升了命中率和搜索结果的实用性。

最后，我们探讨了将根节点优化与子图权重优化策略结合使用的效果。实验结果表明，结合了这两种优化策略的 KeyKG 算法在性能上达到了新的高度。尽管整体运行时间有所增加，但 MP、MR 和 NDCG 等指标均得到了进一步的提升，这充分证明了两种策略结合使用可以更有效地提升 KeyKG 算法在 Mashup 推荐领域的性能。

综上所述，我们的研究验证了根节点优化策略与子图权重优化策略在 KeyKG 算法中的显著优化作用，并证明了将两者结合使用能够进一步提升算法的性能。这些结果不仅证实了两种优化策略的泛化能力，也为后续在 Mashup 推荐领域的研究和应用提供了有价值的参考。具体而言，研究人员可以运用根节点优化策略将连通性大的 API 集合中的节点设定为根节点，同时运用子图权重优化策略，将 API 构成的网络的权重设置为 API 在 Mashup 中共同出现的次数与两 API 节点度乘积的倒数，以实现更优质的推荐效果。

第五章　总结与展望

本文围绕 Mashup 推荐领域的挑战与问题，提出了一种创新的基于静态中心标注法的 Mashup 组合推荐算法 FAR，旨在提高 API 推荐的效率和准确性。通过构建高效的离线索引结构和融入独特的子图评价机制，算法显著提升了推荐系统的性能。同时，柱状搜索策略的运用有效优化了子图生成速度，并避免了过度强调 API 兼容性而忽略实用性的问题。此外，本文还提出了根节点优化策略与子图权重优化策略，为相关领域的研究提供了有益的启示。

尽管本文提出的算法在性能上取得了显著的提升，但仍有进一步优化的空间。例如，可以考虑将动态信息纳入算法中，以增强推荐结果的实时性。此外，探索更为高效的剪枝策略也是提升算法运行效率的重要途径。在未来的工作中，我们将继续深入研究，以期不断优化算法性能，提升其实用性，为用户提供更加高效、精准的 API 推荐服务。

参考文献

[1] NAWAF A. ALMOLHIS M A A. Analysis of End-User Mashup Application in Web applications[J]. Tuijin Jishu/Journal of Propulsion Technology, 2023. DOI: 10.52783/tjjpt.v44.i3.686.

[2] XIE X, ZHANG J, RAMACHANDRAN R, et al. Goal-Driven Context-Aware Service Recommendation for Mashup Development[J]. 2022 IEEE/ACIS 23rd International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2022: 19-26. DOI: 10.1109/SNPD54884.2022.10051805.

[3] BOULAKBECH M, MESSAI N, SAM Y, et al. Configuration approach for personalized travel mashup[J]. Concurrency and Computation: Practice and Experience, 2022, 35. DOI: 10.1002/cpe.6811.

[4] PASHCHENKO D. Fully Remote Software Development as a New Standard in the IT Industry: European Study 2022—2023[J]. Programmnaya Ingeneria, 2023. DOI: 10.17587/prin.14.217-224.

[5] MARTIROSYAN A, OGHLUKYAN A. ECONOMIC IMPACT OF DEVELOPING OF WEB TECHNOLOGIES[J]. ALTERNATIVE, 2022. DOI: 10.55528/18292828-2022.1-26.

[6] GONG W, ZHANG X, CHEN Y, et al. DAWAR: Diversity-aware Web APIs Recommendation for Mashup Creation based on Correlation Graph[J]. Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2022. DOI: 10.1145/3477495.3531962.

[7] ZHOU J, JIANG B J, YANG J, et al. Service Discovery Method Based on Knowledge Graph and Word2vec[J]. Electronics, 2022. DOI: 10.3390/electronics11162500.

[8] SHEN L, WANG Y, ZHANG S, et al. Similarity and Complementarity Attention-Based Graph Neural Networks for Mashup-Oriented Cloud API Recommendation[J]. Electronics, 2023. DOI: 10.3390/electronics12214436.

[9] QI L, HE Q, CHEN F, et al. Data-Driven Web APIs Recommendation for Building Web Applications [J]. IEEE Transactions on Big Data, 2022, 8: 685-698. DOI: 10.1109/TBDATA.2020.2975587.

[10] XIE F, ZHANG Y, PRZYSTUPA K, et al. A Knowledge Graph Embedding Based Service Recommendation Method for Service-Based System Development[J]. Electronics, 2023. DOI: 10.3390/electronics12132935.

[11] WANG X, XI M, YIN J. Functional and Structural Fusion based Web API Recommendations in Heterogeneous Networks[J]. 2023 IEEE International Conference on Web Services (ICWS), 2023: 91-96. DOI: 10.1109/ICWS60048.2023.00025.

[12] XIANG J, CHEN W, WANG Y, et al. Interactive Web API Recommendation for Mashup Development based on Light Neural Graph Collaborative Filtering[J]. 2023 26th International Conference on Computer Supported Cooperative Work in Design (CSCWD), 2023: 1926-1931. DOI: 10.1109/CSCWD57460.2023.10152817.

[13] SANG C, DENG X, LIAO S. Mashup-Oriented Web API Recommendation Via Full-Text Semantic Mining of Developer Requirements[J]. IEEE Transactions on Services Computing, 2023, 16: 2755-

2768. DOI: 10.1109/TSC.2023.3245652.

[14] WANG Y, XIANG J, CHENG H, et al. Towards Dynamic Evolutionary Analysis of ProgrammableWeb for API-Mashup Ecosystem[C]//2023 26th International Conference on Computer Supported Cooperative Work in Design (CSCWD). 2023: 1716-1721. DOI: 10.1109/CSCWD57460.2023.10152769.

[15] GAO W, WU J. Multi-Relational Graph Convolution Network for Service Recommendation in Mashup Development[J]. Applied Sciences, 2022. DOI: 10.3390/app12020924.

[16] GU Q, CAO J, LIU Y. CSBR: A Compositional Semantics-Based Service Bundle Recommendation Approach for Mashup Development[J]. IEEE Transactions on Services Computing, 2022, 15: 3170-3183. DOI: 10.1109/TSC.2021.3085491.

[17] NAWAZ M, KHAN S U R, AHMAD B, et al. CAPIRS: COVID-19-Based Application Programming Interface Recommendation System for the Developers[J]. J. Inf. Knowl. Manag., 2022, 21: 2240004:1-2240004:30. DOI: 10.1142/s0219649222400044.

[18] GONG W, LV C, DUAN Y, et al. Keywords-driven web APIs group recommendation for automatic app service creation process[J]. Software: Practice and Experience, 2020, 51: 2337-2354. DOI: 10.1002/spe.2902.

[19] YIN Y, HUANG Q, GAO H, et al. Personalized APIs Recommendation With Cognitive Knowledge Mining for Industrial Systems[J]. IEEE Transactions on Industrial Informatics, 2021, 17: 6153-6161. DOI: 10.1109/TII.2020.3039500.

[20] GONG W, ZHANG W, BILAL M, et al. Efficient Web APIs Recommendation With Privacy-Preservation for Mobile App Development in Industry 4.0[J]. IEEE Transactions on Industrial Informatics, 2022, 18: 6379-6387. DOI: 10.1109/tii.2021.3133614.

[21] CHEN J, WANG Y, HUANG Q, et al. Open APIs recommendation with an ensemble-based multi-feature model[J]. Expert Systems with Applications, 2022, 196: 116574-.

[22] WU H, DUAN Y, YUE K, et al. Mashup-Oriented Web API Recommendation via Multi-Model Fusion and Multi-Task Learning[J]. IEEE Transactions on Services Computing, 2022, 15(6): 3330-3343. DOI: 10.1109/TSC.2021.3098756.

[23] KOU H, XU J, QI L. Diversity-driven automated web API recommendation based on implicit requirements[J]. Applied Soft Computing, 2023.

[24] QI L, LIN W, ZHANG X, et al. A Correlation Graph Based Approach for Personalized and Compatible Web APIs Recommendation in Mobile APP Development[J]. IEEE Transactions on Knowledge and Data Engineering, 2023, 35: 5444-5457. DOI: 10.1109/TKDE.2022.3168611.

[25] XU Y, DING Y, JIANG Z, et al. Web APIs recommendation with neural content embedding for mobile multimedia computing[J]. WIRELESS NETWORKS, 2023.

[26] SUN Y, XIAO X, CUI B, et al. Finding group Steiner trees in graphs with both vertex and edge weights[J]. Proceedings of the VLDB Endowment, 2021.

[27] MATIJEVIĆ L, JELIĆ S, DAVIDOVIĆ T. General variable neighborhood search approach to group steiner tree problem[J]. Optimization Letters, 2023, 17(9): 2087-2111.

[28] YANG S, SUN Y, LIU J, et al. Approximating Probabilistic Group Steiner Trees in Graphs[J]. Proc.

VLDB Endow., 2022, 16: 343-355. DOI: 10.14778/3565816.3565834.

[29] KORTSARZ G, NUTOV Z. The minimum degree Group Steiner problem[J]. Discret. Appl. Math., 2022, 309: 229-239. DOI: 10.1016/j.dam.2021.12.003.

[30] GE Y, CHEN Z, KONG W, et al. An Efficient Dynamic Programming Algorithm for Finding Group Steiner Trees in Temporal Graphs[J]. International Journal of Intelligent Systems, 2023. DOI: 10.11 55/2023/1974161.

[31] QIU Z M, ZHAO H H, YANG J. A group decision making approach based on the multi-dimensional Steiner point[J]. AIMS Mathematics, 2024, 9(1): 942-958.

[32] 范国栋, 李静, 祝铭, 等. 图数据库中有向二分图构建的服务组合[J]. 重庆大学学报：自然科学版, 2020, 43(7): 11.

[33] JIANG B, YANG J, QIN Y, et al. A Service Recommendation Algorithm Based on Knowledge Graph and Collaborative Filtering[J]. IEEE Access, 2021, 9: 50880-50892. DOI: 10.1109/ACCESS.2021.3 068570.

[34] SHI M, ZHUANG Y, TANG Y, et al. Web Service Network Embedding Based on Link Prediction and Convolutional Learning[J]. IEEE Transactions on Services Computing, 2021, 15: 3620-3633. DOI: 10.1109/TSC.2021.3103481.

[35] CHEN H, WU H, LI J, et al. Keyword-Driven Service Recommendation Via Deep Reinforced Steiner Tree Search[J]. IEEE Transactions on Industrial Informatics, 2023, 19: 2930-2941. DOI: 10.1109/T II.2022.3177411.

[36] ZHANG Y, YU J X. Hub labeling for shortest path counting[C]//Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. 2020: 1813-1828.

[37] FENG Q, PENG Y, ZHANG W, et al. Towards real-time counting shortest cycles on dynamic graphs: A hub labeling approach[C]//2022 IEEE 38th International Conference on Data Engineering (ICDE). 2022: 512-524.

[38] BLUM J, STORANDT S. Customizable Hub Labeling: Properties and Algorithms[C]//International Computing and Combinatorics Conference. 2022: 345-356.

[39] LI W, QIAO M, QIN L, et al. Distance labeling: on parallelism, compression, and ordering[J]. The VLDB Journal, 2022, 31(1): 129-155.

[40] OUYANG D, WEN D, QIN L, et al. When hierarchy meets 2-hop-labeling: efficient shortest distance and path queries on road networks[J]. The VLDB Journal, 2023, 32(6): 1263-1287.

[41] SHI Y, CHENG G, KHARLAMOV E. Keyword search over knowledge graphs via static and dynamic hub labelings[C]//Proceedings of The Web Conference 2020. 2020: 235-245.

[42] 凌春阳, 邹艳珍, 林泽琦, 等. 基于图嵌入的软件项目源代码检索方法[J]. 软件学报, 2019, 30(5): 1481-1497.